NOVA RUN DESIGN DOCUMENT

PROCEDURAL SIDE-SCROLLING SHOOTER

VISTA SCIENCES
Marc King & Jeff Wright
2016-11-04

[CIS-487-001 FALL 2016 • UNIVERSITY OF MICHIGAN – DEARBORN • DR. BRUCE MAXIM]

Project Description

This game design document describes the details of a 2D game created using the Unity engine for the CIS-487 Game Design I course with Dr. Bruce Maxim during the Fall 2016 semester at the University of Michigan – Dearborn. The project will be to design and implement a side-scrolling shooter with procedurally generated levels.

Version History

Version #	Implemented By	Revision Date	Approved By	Approval Date	Reason
1.0	Marc & Jeff	2016-10-03	Marc & Jeff	2016-10-03	Initial Version
1.1	Marc	2016-10-05	Jeff	2016-10-10	Tricks Template
1.2	Jeff	2016-10-10	Marc	2016-10-11	Filled in sections
1.3	Marc	2016-10-15	Jeff	2016-10-18	Characters, Story, Story Progression, Gameplay
1.4	Marc	2016-10-18	Jeff	2016-10-18	Added gameplay diagrams.
1.5	Marc	2016-10-21	Jeff	2016-10-29	Revised sections based on peer feedback: added clearer description of player's weapons capabilities, obstacle movement, hardware requirements, and algorithm styles.
1.5	Marc	2016-10-22	Jeff	2016-10-29	Added game flow diagram, more detail on obstacle lifetime.
1.6	Jeff	2016-10-29	Marc	2016-11-02	Added details to Art Style section
1.7	Marc	2016-11-02	Jeff	2016-11-03	Updated gameplay, and music, and algorithms.

Table of Contents

1.0 Characters

Captain Justice Roberts

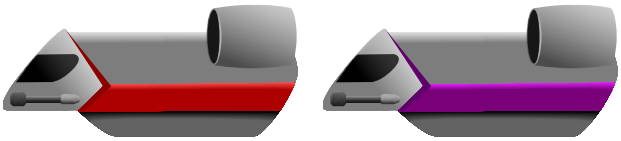
Our main character is a retired space force pilot that spends their time transporting tourists to the corona of Sol.

Keplerian Fighters

Keplerian fighters are agile and possess little in the way of fire power. While easy to destroy, they swarm in vast numbers.

Keplerian Warships

Bulky and slow to react, Keplerian warships plod their course while laying heavy fire in their path.



2.0 Story

It's another boring day for Captain Justice Roberts, taking another boring batch of spoiled tourists to feign just-the-right-level-of-awe for their peers at the edges of our sun's corona. The tedium is broken by an urgent message on subspace frequencies.

Recent negotiations with the Keplerians have broken down and war has been declared. Known for their viciousness, the Keplerian's first move is to force our sun into its red giant phase.

While the sun rapidly expands to engulf the inner solar system, humanity's only hope is to rush with reckless abandon to rendezvous at Pluto Base. Caught near the epicenter of the growing nuclear annihilation, Captain Roberts faces a perilous dash toward the edge of the solar system.

2.1 Theme

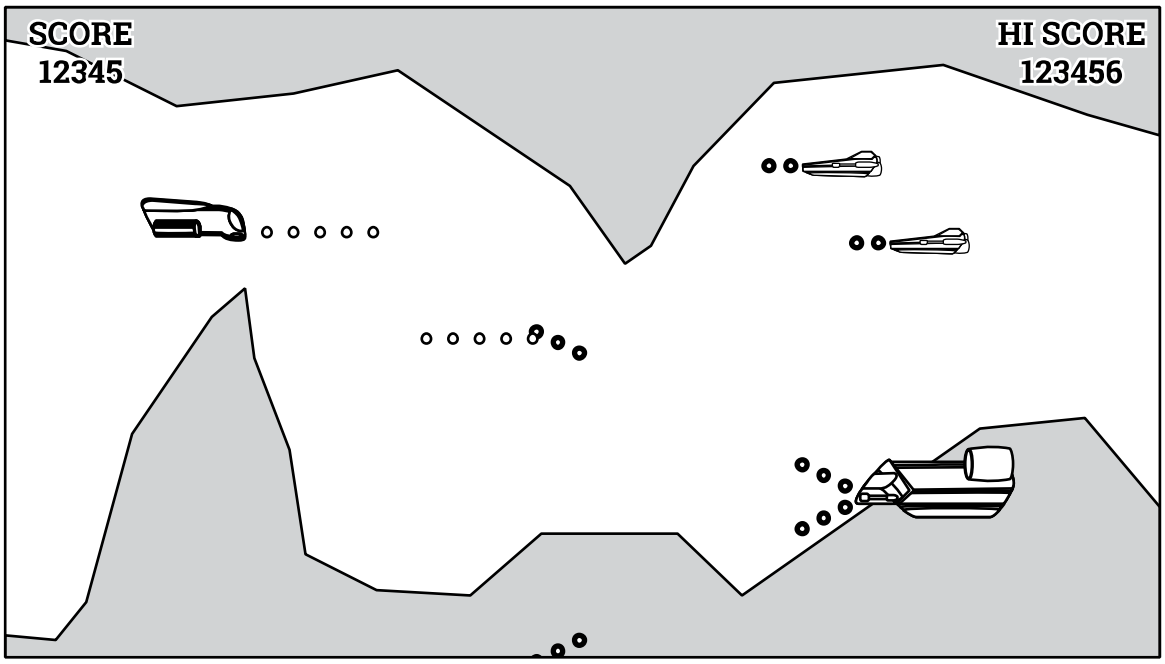
This game is about a hopeless rush to safety. It continues to become more frantic as it progresses. Eventually it reaches a level of difficulty that is impossible for a human to overcome.

3.0 Story Progression

The game begins shortly after Captain Roberts receives a warning about our sun's imminent expansion. Roberts immediately begins heading towards Pluto Base as quickly as possible. As the Keplerians flood the inner solar system to prevent Earth forces from escaping, Roberts encounters increasingly numerous hostile forces.

4.0 Gameplay

Basic gameplay consists of the player's ship flying to the right while having to dodge terrain obstacles, enemy ships, and weapons fire.



4.1 Goals

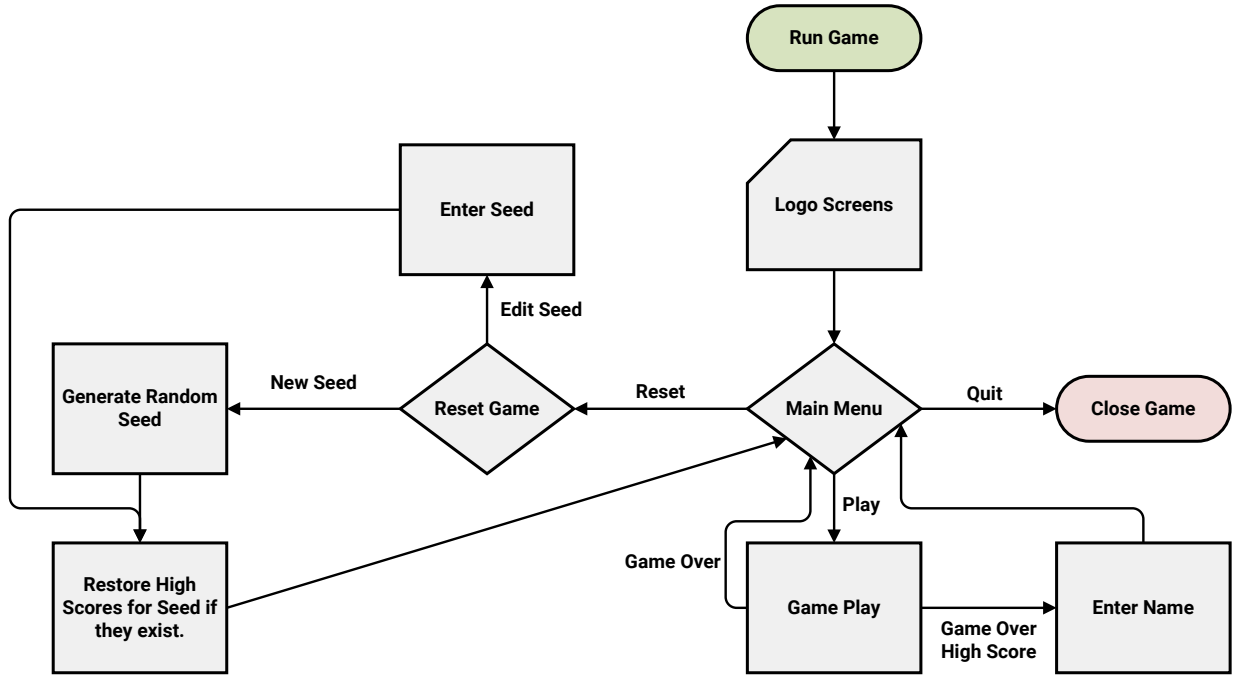
The primary goal of the player is to gain recognition through achieving a high score. Player's score points by defeating enemies and for the distance they travel in a play-through. The game keeps track of the highest five scores, and associated names, for each game seed. Although games will be randomly generated, players will be competing for high score within the same procedurally generated games.

4.2 User Skills

1. Keyboard or game controller
2. Reflexes
3. Strategy
4. Timing

4.3 Game Mechanics

4.3.1 Overall Flow of Game



4.3.2 Player Movement

Players can move their ship using the cursor keys or WASD keys, and fire their weapons using the space bar or mouse button. An aiming cursor is provided that can be position with the mouse. Firing the player's weapon results in losing a small amount of score, this is promote burst fire instead of firing contantly.

The players ship travels in different directions at different speeds. If turning (going up or down) is considered the base speed, then going forward is 150% the speed of turning. The speed of moving backwards is dependent on how fast the obstacles are currently moving; i.e. the ship will travel backwards as fast as necessary to always move forward at 0.25 units per second relative to 'stationary' obstacles. This mechanic allows for the player to be limited in their backwards motion, but not be completely hindered by it when obstacles are moving very quickly.

The player's ship fires its weapon horizontally from the front of the ship. Players are able to hold the space bar to continuously fire, or to press the space bar repeatedly. In either case, the player's ship cannot fire more than once every 0.25 seconds.

4.3.3 Obstacles

Obstacles are pieces of landscape that the player must avoid. A collision between the player and an obstacle will result in the destruction of the player. Obstacles continuously move from the right side of the screen to the left side, but in the context of the game they are stationary. The leftward movement is meant to simulate a continuous motion of the player toward the right.

4.3.3.1 StationaryObstacle

The StationaryObstacle prefab is the primary GameObject used for managing obstacles. It is 12 units by 12 units in size, and consists of a Box Collider 2D, a Rigidbody 2D, and the StationaryObstacleMover script. Upon instantiation, the StationaryObstacle will also have a child GameObject created within it that consists of the Sprite Renderer, a Rigidbody 2D, and a Polygon Collider 2D. This Polgygon Collider 2D is what the player must actually avoid and is based on the graphics of the sprite.

All obstacles continually travel to the left at the same speed (to simulate a constant rightward motion of the player ship). All of the obstacles continue to increase their speed at the same constant rate. Any contact with an obstacle by the player's ship results in a game over.

Obstacles begin with a velocity of 1 unit per second, and will increase their velocity to meet a target velocity of 10 units per second within 3 minutes. Based on our rough prototype, once obstacles are moving at 15-20 units per second, it becomes unrealistic for average human response time overcome. This gives a typical play-through a length of 3-5 minutes; which should be long enough to be engaging, and short enough to promote taking turns with other players.

4.3.4 Enemies

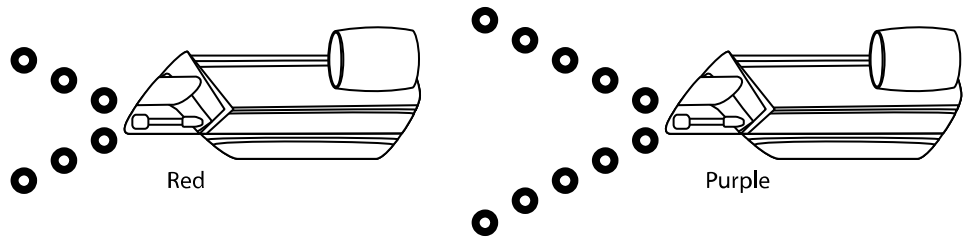
Enemies will travel to the left at a rate slightly faster than obstacles. They will fire weapons at consistent rates and any contact with the enemy ships by the player will result in a game over. Contact by the player with enemy weapon fire results in a reduction of the player's shields. Enemies do not interact with obstacles.

Fighters

Fighters will move 25% faster than obstacles, and will fire in a straight line once every second. Red fighters are destroyed with a single weapon hit from the player, and shoot 2 blasts. Purple fighters are destroyed with two weapon hits from the player, and shoot 3 blasts.

Warships

Warships will move 10% faster than obstacles, and will fire two streams at 30º once every 3 seconds. Red warships are destroyed by 3 weapon hits from the player, and shoot 3 blasts in each direction. Purple warships are destroyed by 4 weapon hits from the player, and shoot 5 blasts in each direction.



4.4 Progression and Challenge

As the player progresses through each level, the scrolling speed will increase. Enemies will become more numerous or have upgraded weapons. The player does not receive any upgrades. This is done to ensure that the game will reach an end.

4.5 Procedural Generation

To procedurally generate the levels, there will exist game objects off the right side of the camera that will create obstacle objects. When an obstacle object leaves the creator

object's box collider the creator will randomly instantiate the next obstacle and place it next to the previous obstacle. All existing obstacles will update their velocity from a master game object every physics tick. After obstacles completely leave from view of the camera, they are destroyed.

4.6 Losing

The player loses when their ship is destroyed. This can happen by either hitting the terrain on the top or bottom of the screen or any floating obstacles. The player can also be destroyed by colliding with enemy ships.

When the player's ship is destroyed a screen will be displayed with their score from that play through of the level. If they have achieved a high score they will be prompted for their name.

5.0 Art Style

This game is a procedural loading 2D side-scrolling shooter. All of the graphics in the game are cell shaded and were created using Adobe Illustrator. The background graphics are layered and parallax is used to give the illusion of 3D depth. The levels background pieces are 1200px X 1200px with 150px extending off the top and bottom of the screen to account for any differences in screen resolutions.

The players ship is visually different than the enemy ships. The players ship is 100px X 50px and is designed to look like a transport ship.

Enemy ships are also visually different depending on the type of weapon that they use. Enemy fighters are 75px X 50px and are designed to look nimble and quick. Enemy warships are 150px X 75px and are designed to look bulky and powerful.

Enemy ships come in two different difficulties. Enemies with red stripes are standard and enemies with purple stripes are more powerful. Purple enemies take more hits to destroy and fire more blasts.

6.0 Music and Sounds

The music was composed by Jesse Lee Mason exclusively for Nova Run, and was composed using an acoustic guitar recorded on multiple tracks. It consists of a song that plays through the opening logo, the main menu, and the introduction. Four songs have the potential to play during a play through, but since they are always played in the same order the odds of hearing them all is small. A slightly upbeat song plays during the game over screen in an attempt to make player deaths somewhat less frustrating.

Since sound cannot propagate in the vacuum of space, we felt it was inappropriate to have sounds of explosions of gun fire. However, to provide some level of aural feedback

to players when they destroy enemies or die, we had Jesse record guitar chords and riffs to play when those events occur. Additionally, at the beginning of the game the explosion of the sun is accompanied by the sound of an out-of-tune, slapped string that was slowed down to 20% of its original speed.

7.0 Technical description

The following tools will be used for various aspects of the planning and development of Nova Run.

7.1 Communication

Most communication will occur on the Vista Sciences Slack (vistasciences.slack.com), but text messaging will be used for quick meeting changes and confirmations.

7.2 Planning

Planning of this project will be collected using Microsoft Word documents stored on OneDrive, with some diagrams being created in Microsoft Visio.

7.3 Development

Unity will be the primary development environment, with Visual Studio Code used as the C# script editor, and Bitbucket used as the concurrent versioning system.

7.4 Art & Sound

Art assets will be created in Adobe Illustrator and Adobe Photoshop, and sound and music will be edited using Audacity.

7.5 Platforms

This game will initially be made for Windows OS, but could be ported to Mac OS, Linux, iOS, and Android with relatively little work due to using Unity and the nature of the gameplay controls.

7.6 Hardware Requirements

The expected hardware requirements for Nova Run is any computer capable of running Windows 7. No special graphics hardware will be necessary; as integrated graphics should suffice.

7.7 Algorithms

The closest thing to a special algorithm Nova Run will use is procedural generation using a random number generator with a preset seed. To allow random number generation to be reproducible, we used the System.Random object instead of Unity's Random object. This allowed us to keep two separate instances of the Random objects;

one for enemies and one for obstacles. Two separate instances were required to ensure reproducibility because Unity cannot ensure the order of execution of Update() or FixedUpdate() methods on various GameObjects. Due to the nature of the setup of obstacles and enemies, we could ensure that those systems asked for random numbers in a consistent manner individually, but together there was the risk of timing issues affecting the order of random calls.

To produce reproducibly random enemies that also increased in difficulty over time, the probability of an enemy being purple was first determined by establishing a base probability of a purple enemy and then increasing that probability over time until it eventually reached 1.0. The same calculation was performed to determine if a enemy would be a small enemy or a large enemy. Tweaking the values in these calculations eventually lead to a system that would produce mostly small red enemies towards the beginning of the game, and as the game progresses enemies at purple and/or large more often until eventually they are all large and purple.